SimplicialSurfaces

Computation with simplicial surfaces and folding processes.

0.1

07/07/2017

Alice Niemeyer

Markus Baumeister

Alice Niemeyer
Email: Alice.Niemeyer@Mathb.RWTH-Aachen.De
Homepage: https://www.mathb.rwth-aachen.de/Mitarbeiter/niemeyer.php
Address: Alice Niemeyer
Lehrstuhl B für Mathematik
RWTH Aachen
Pontdriesch 10/16
52062 Aachen
GERMANY

Markus Baumeister
Email: markus.baumeister1@rwth-aachen.de
Homepage: https://www.mathb.rwth-aachen.de/Mitarbeiter/baumeister.php
Address: Markus Baumeister
Lehrstuhl B für Mathematik
RWTH Aachen
Pontdriesch 10/16
52062 Aachen
GERMANY

Contents

1 Getting started **3**
- 1.1 What can it do? . . . 3
- 1.2 Playing with simplicial surfaces . . . 3

2 Simplicial Surfaces **4**
- 2.1 Constructors for Simplicial Surfaces . . . 5
- 2.2 Access to the incidence structure . . . 8
- 2.3 Basic properties of simplicial surfaces . . . 10
- 2.4 Functions for simplicial surfaces . . . 13
- 2.5 Advanced properties of simplicial surfaces . . . 14
- 2.6 Local and global orientations . . . 16
- 2.7 Technical functions (for development) . . . 19

3 Wild Simplicial Surfaces **22**
- 3.1 Constructors for wild simplicial surfaces . . . 22
- 3.2 Attributes and properties of wild coloured simplicial surfaces . . . 24
- 3.3 Functions for wild coloured simplicial surfaces . . . 27

Index **29**

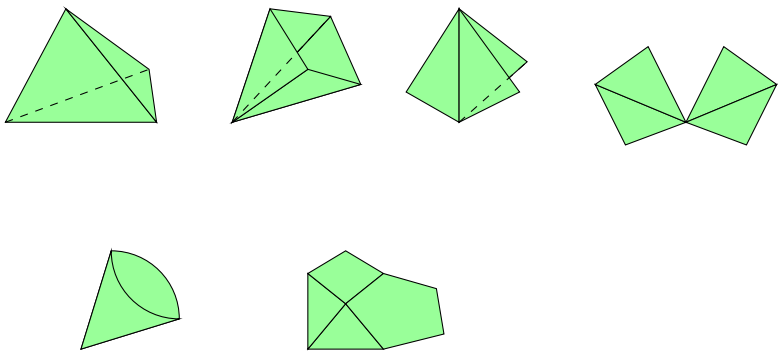
Chapter 1

Getting started

1.1 What can it do?

The SimplicialSurface-package contains this basic functionality:

1. Working with simplicial surfaces (and generalisations of that), for example:



2. Working with edge colourings of simplicial surfaces (in general and for the purpose of an embedding).

3. Treatment of folding and unfolding for these objects.

To use this package, you have to load it into GAP via

Example

```
gap> LoadPackage("SimplicialSurfaces");
true
```

1.2 Playing with simplicial surfaces

Since the platonic solids are pre-defined we use them to show a few capabilities of this package.

Chapter 2

Simplicial Surfaces

A SimplicialSurface-object in GAP represents mathematical objects that are a generalization of simplicial surfaces. On the most basic level it consists of vertices, edges and faces, together with an incidence relation between them. This information is saved in the following form:

- the vertices are represented by a set of positive integers
- the edges are represented by a set of positive integers
- the faces are represented by a set of positive integers
- the incidence relation between vertices and edges (that is, which vertices lie in which edges) is represented by a list VerticesOfEdges. For each edge e the entry VerticesOfEdges[e] is a set of all vertices that are incident to the edge e.
- the incidence relation between edges and faces (that is, which edges lie in which faces) is represented by a list EdgesOfFaces. For each face f the entry EdgesOfFaces[f] is a set of all edges that are incident to the face f.

Every other incidence (like VerticesOfFaces or EdgesOfVertices) is represented in an analogous fashion. Furthermore we impose some restrictions onto these incidence relations:

- Every edge is incident to exactly two vertices
- Every face is incident to the same number of vertices and edges (at least three). Additionally these are cyclically oriented (to represent an n-gon)
- The incidence relations are transitive
- Every vertex and every edge is incident to at least one face

Note that it is allowed for an edge to be incident to more than two faces. In addition it is sometimes necessary to distinguish between the two sides of a single face. If this is irrelevant to your application, you can completely ignore this (it will be handled in the background). To use this distinction each side of a face f gets a name - the defaults are +f and -f (but you can give them custom names if you want to). To distinguish which side is which (not only by a name, but geometrically) we save a cyclic ordering of the vertices (or edges) of each face which we associate with the side +f. (In an embedding to $\mathbb{R}^3$ we could associate the cyclic ordering with the normal vector of the face that is defined by the right-hand-rule from physics.)

2.1 Constructors for Simplicial Surfaces

2.1.1 Janushead (for)

▷ `Janushead()` (operation)

Returns: a simplicial surface

Return a simplicial surface that represents a janus head.

2.1.2 Tetrahedron (for)

▷ `Tetrahedron(`*arg*`)` (operation)

Returns: a simplicial surface

Return a simplicial surface that represents a tetrahedron.

2.1.3 Cube (for)

▷ `Cube(`*arg*`)` (operation)

Returns: a simplicial surface

Return a simplicial surface that represents a cube

2.1.4 Octahedron (for)

▷ `Octahedron(`*arg*`)` (operation)

Returns: a simplicial surface

Return a simplicial surface that represents a octahedron.

2.1.5 Dodecahedron (for)

▷ `Dodecahedron(`*arg*`)` (operation)

Returns: a simplicial surface

Return a simplicial surface that represents a dodecahedron.

2.1.6 Icosahedron (for)

▷ `Icosahedron(`*arg*`)` (operation)

Returns: a simplicial surface

Return a simplicial surface that represents a icosahedron.

2.1.7 SimplicialSurfaceByUpwardIncidence (for IsSet, IsSet, IsSet, IsList, IsList)

▷ `SimplicialSurfaceByUpwardIncidence(`*vertices, edges, faces, edgesOfVertices, facesOfEdges*`)` (operation)

▷ `SimplicialSurfaceByUpwardIncidenceNC(`*vertices, edges, faces, edgesOfVertices, facesOfEdges*`)` (operation)

Returns: a simplicial surface

This constructor of a simplicial surface uses the following information:

- The set of vertices (alternatively a positive integer n, which will be interpreted as the set [1..n])
- The set of edges (alternatively a positive integer n, which will be interpreted as the set [1..n])

- The set of faces (alternatively a positive integer n, which will be interpreted as the set [1..n])
- The list edgesOfVertices. For each vertex v the entry edgesOfVertices[v] is a set of all edges that are incident to the vertex v.
- The list facesOfEdges. For each edge e the entry facesOfEdges[e] is a set of all faces that are incident to the edge e.

In this constructor there is no way of distinguishing the sides of each face, so this selection will be made randomly. The NC-version does not check whether the given input follows the following criteria:

- vertices, edges and faces have to be either positive integers or sets of positive integers.
- For each vertex v the entry edgesOfVertices[v] has to be a subset of the set of edges.
- Every edge has to be in one of the sets in the list edgesOfVertices.
- For each edge e the entry facesOfEdges[e] has to be a subset of the set of faces.
- Each face has to be in one of the sets in the list facesOfEdges.

2.1.8 SimplicialSurfaceByDownwardIncidence (for IsSet, IsSet, IsSet, IsList, IsList)

▷ `SimplicialSurfaceByDownwardIncidence(`*vertices, edges, faces, verticesOfEdges, edgesOfFaces*`)` (operation)
▷ `SimplicialSurfaceByDownwardIncidenceNC(`*vertices, edges, faces, verticesOfEdges, edgesOfFaces*`)` (operation)

Returns: a simplicial surface

This constructor of a simplicial surface uses the following information:

- The set of vertices (alternatively a positive integer n, which will be interpreted as the set [1..n])
- The set of edges (alternatively a positive integer n, which will be interpreted as the set [1..n])
- The set of faces (alternatively a positive integer n, which will be interpreted as the set [1..n])
- The list verticesOfEdges. For each edge e the entry verticesOfEdges[e] is a set of all vertices that are incident to the edge e.
- The list edgesOfFaces. For each face f the entry edgesOfFaces[f] is a set of all edges that are incident to the face f.

This constructor does not distinguish different sides of each face, so this selection will be made randomly. The NC-version does not check whether the given input follows the following criteria:

- vertices, edges and faces have to be either positive integers or sets of positive integers.
- For each edge e the entry verticesOfEdges[e] has to be a subset of the set of vertices.
- Every vertex has to be in one of the sets in the list verticesOfEdges.
- For each face f the entry edgesOfFaces[f] has to be a subset of the set of edges.
- Each edge has to be in one of the sets in the list edgesOfFaces.

2.1.9 SimplicialSurfaceByDownwardIncidenceWithOrientation (for IsSet, IsSet, IsSet, IsList, IsList)

▷ SimplicialSurfaceByDownwardIncidenceWithOrientation(*vertices, edges, faces, verticesOfEdges, edgesOfFaces[, namesOfFaces]*) (operation)
▷ SimplicialSurfaceByDownwardIncidenceWithOrientationNC(*vertices, edges, faces, verticesOfEdges, edgesOfFaces[, namesOfFaces]*) (operation)

Returns: a simplicial surface

This constructor of a simplicial surface uses the following information:

- The set of vertices (alternatively a positive integer n, which will be interpreted as the set [1..n])
- The set of edges (alternatively a positive integer n, which will be interpreted as the set [1..n])
- The set of faces (alternatively a positive integer n, which will be interpreted as the set [1..n])
- The list verticesOfEdges. For each edge e the entry verticesOfEdges[e] is a set of all vertices that are incident to the edge e.
- The list edgesOfFaces. For each face f the entry edgesOfFaces[f] is a list of all edges that are incident to the face f. This list has to have the property that two adjacent edges in this list (where we count the first and the last entry to be adjacent) have one vertex in common (this is always the case if the face is a triangle).
- The optional argument namesOfFaces. This is a list and for each face f it has an entry namesOfFaces[f] that is a list with exactly two elements. The first element will be the name of the primary side of the face, the second element will be the name of the secondary side of the face. If this argument is not given, the default naming scheme (+f for primary and -f for secondary) is used.

The ordering of the edges in the list edgesOfFaces defines which side of the the simplicial surface will consider as primary. The NC-version does not check whether the given input follows the following criteria:

- vertices, edges and faces have to be either positive integers or sets of positive integers.
- For each edge e the entry verticesOfEdges[e] has to be a subset of the set of vertices.
- Every vertex has to be in one of the sets in the list verticesOfEdges.
- For each face f the entry edgesOfFaces[f] has to be a subset of the set of edges.
- Each edge has to be in one of the sets in the list edgesOfFaces.
- The edge lists in the components of edgesOfFaces conform to the adjacency condition from before.
- For each face f the list namesOfFaces[f] has exactly two elements that are both integers.

2.1.10 SimplicialSurfaceByVerticesInFaces (for IsSet, IsSet, IsList)

▷ SimplicialSurfaceByVerticesInFaces(*vertices, faces, verticesOfFaces[, namesOfFaces]*) (operation)
▷ SimplicialSurfaceByVerticesInFacesNC(*vertices, faces, verticesOfFaces[, namesOfFaces]*) (operation)

Returns: a simplicial surface
This constructor of a simplicial surface uses the following information:

- The set of vertices (alternatively a positive integer n, which will be interpreted as the set [1..n])
- The set of faces (alternatively a positive integer n, which will be interpreted as the set [1..n])
- The list verticesOfFaces. For each face f the entry verticesOfFaces[f] is a set of all vertices that are incident to the face f.
- The optional argument namesOfFaces. This is a list and for each face f it has an entry namesOfFaces[f] that is a list with exactly two elements. The first element will be the name of the primary side of the face, the second element will be the name of the secondary side of the face. If this argument is not given, the default naming scheme (+f for primary and -f for secondary) is used.

We assume that two vertices that are adjacent in a list verticesOfFaces[f] (where the first and the last entry are adjacent) will have an edge between them. These edges will be constructed automatically. This constructor does not distinguish different sides of each face, so this selection will be made randomly. The NC-version does not check whether the given input follows the following criteria:

- vertices and faces have to be either positive integers or sets of positive integers.
- For each face f the entry verticesOfFaces[f] has to be a subset of the set of vertices.
- For each face f the list verticesOfFaces[f] has to contain at least three different vertices.
- For each face f the list namesOfFaces[f] has exactly two elements that are both integers.

2.2 Access to the incidence structure

2.2.1 Vertices (for IsSimplicialSurface)

▷ Vertices(*simpSurf*) (operation)
▷ VerticesAttributeOfSimplicialSurface(*simpSurf*) (attribute)

Returns: the set of vertices, a set of positive integers
Returns the vertices as a set. The vertices are positive integers.

2.2.2 Edges (for IsSimplicialSurface)

▷ Edges(*simpSurf*) (attribute)

Returns: the set of edges, a set of positive integers
Returns the edges as a set. The edges are positive integers.

2.2.3 Faces (for IsSimplicialSurface)

▷ `Faces(`*simpSurf*`)` (attribute)

Returns: the set of faces, a set of positive integers

Returns the faces as a set. The faces are positive integers.

2.2.4 NrOfVertices (for IsSimplicialSurface)

▷ `NrOfVertices(`*simpSurf*`)` (attribute)

Returns: a non-negative integer

Returns the number of vertices.

2.2.5 NrOfEdges (for IsSimplicialSurface)

▷ `NrOfEdges(`*simpSurf*`)` (attribute)

Returns: a non-negative integer

Returns the number of edges.

2.2.6 NrOfFaces (for IsSimplicialSurface)

▷ `NrOfFaces(`*simpSurf*`)` (attribute)

Returns: a non-negative integer

Returns the number of faces.

2.2.7 EdgesOfVertices (for IsSimplicialSurface)

▷ `EdgesOfVertices(`*simpSurf*`)` (attribute)

Returns: a list of sets of positive integers

Return a list edgesOfVertices such that

- if v is a vertex then edgesOfVertices[v] is the set of all edges that are incident to v.
- if v is not a vertex then edgesOfVertices[v] is not bound.

2.2.8 FacesOfVertices (for IsSimplicialSurface)

▷ `FacesOfVertices(`*simpSurf*`)` (attribute)

Returns: a list of sets of positive integers

Return a list facesOfVertices such that

- if v is a vertex then facesOfVertices[v] is the set of all faces that are incident to v.
- if v is not a vertex then facesOfVertices[v] is not bound.

2.2.9 VerticesOfEdges (for IsSimplicialSurface)

▷ `VerticesOfEdges(`*simpSurf*`)` (attribute)

Returns: a list of sets of positive integers

Return a list verticesOfEdges such that

- if e is an edge then verticesOfEdges[e] is the set of all vertices that are incident to e.

- if e is not an edge then verticesOfEdges[e] is not bound.

2.2.10 FacesOfEdges (for IsSimplicialSurface)

▷ `FacesOfEdges(simpSurf)` (attribute)

Returns: a list of sets of positive integers

Return a list facesOfEdges such that

- if e is an edge then facesOfEdges[e] is the set of all faces that are incident to e.
- if e is not an edge then facesOfEdges[e] is not bound.

2.2.11 VerticesOfFaces (for IsSimplicialSurface)

▷ `VerticesOfFaces(simpSurf)` (attribute)

Returns: a list of sets of positive integers

Return a list verticesOfFaces such that

- if f is a face then verticesOfFaces[f] is the set of all vertices that are incident to v.
- if f is not a face then verticesOfFaces[f] is not bound.

2.2.12 EdgesOfFaces (for IsSimplicialSurface)

▷ `EdgesOfFaces(simpSurf)` (attribute)

Returns: a list of sets of positive integers

Return a list edgesOfFaces such that

- if f is a face then edgesOfFaces[f] is the set of all edges that are incident to v.
- if f is not a face then edgesOfFaces[f] is not bound.

2.2.13 EdgeInFaceByVertices (for IsSimplicialSurface, IsPosInt, IsList)

▷ `EdgeInFaceByVertices(simpSurf, face, vertexList)` (operation)

Returns: the edge

Return the edge of the given face that is incident to the given vertices.

2.3 Basic properties of simplicial surfaces

2.3.1 EulerCharacteristic (for IsSimplicialSurface)

▷ `EulerCharacteristic(simpSurf)` (attribute)

Returns: an integer, the Euler characteristic.

Return the Euler characteristic of the given simplicial surface. The Euler characteristic is $|V| - |E| + |F|$, where $|V|$ is the number of vertices, $|E|$ is the number of edges and $|F|$ is the number of faces.

2.3.2 IsTriangleSurface (for IsSimplicialSurface)

▷ `IsTriangleSurface(`*simpSurf*`)` (property)

Returns: `true` or `false`

The property IsTriangleSurface is true if all faces of the simplicial surface are triangles (i.e. they consist of three edges).

2.3.3 IsPathConnected (for IsSimplicialSurface)

▷ `IsPathConnected(`*simpSurf*`)` (property)

Returns: `true` or `false`

If the simplicial surface is path connected (i.e. if every two faces are connected by a face-edge-path) this method returns true, otherwise false.

2.3.4 PathConnectedComponents (for IsSimplicialSurface)

▷ `PathConnectedComponents(`*simpSurf*`)` (attribute)
▷ `PathConnectedComponentOfFace(`*simpSurf, face*`)` (operation)
▷ `PathConnectedComponentOfFaceNC(`*simpSurf, face*`)` (operation)

Returns: a list of simplicial surfaces

Return a list of all path-connected components of the simplicial surface.

If a face is given additionally the path-connected component of this face is returned. The NC-version does not check if the given face actually lies in the simplicial surface.

2.3.5 IsConnected (for IsSimplicialSurface)

▷ `IsConnected(`*simpSurf*`)` (property)

Returns: `true` or `false`

Return if a simplicial surface is connected. If two faces share at least one vertex they are considered to be connected.

2.3.6 ConnectedComponentsAttributeOfSimplicialSurface (for IsSimplicialSurface)

▷ `ConnectedComponentsAttributeOfSimplicialSurface(`*simpSurf*`)` (attribute)
▷ `ConnectedComponents(`*arg*`)` (operation)
▷ `ConnectedComponentOfFace(`*simpSurf, face*`)` (operation)
▷ `ConnectedComponentOfFaceNC(`*simpSurf, face*`)` (operation)

Returns: a list of simplicial surfaces

Return a list of all connected components of the simplicial surface.

If a face is given additionally the connected component of this face is returned. The NC-version does not check if the given face actually lies in the simplicial surface.

2.3.7 UnsortedDegrees (for IsSimplicialSurface)

▷ `UnsortedDegrees(`*simpSurf*`)` (attribute)

Returns: a list of positive integers

Return a list unsortedDegrees with the following property: For each vertex v the entry unsortedDegrees[v] contains the number of faces that are incident to that vertex (this is called the degree of the vertex).

2.3.8 SortedDegrees (for IsSimplicialSurface)

▷ `SortedDegrees(`*simpSurf*`)` (attribute)

Returns: a sorted list of positive integers

Return a sorted list sortedDegrees such that the degree of each vertex (which is defined as the number of incident faces) is contained in the list (counting repetitions).

2.3.9 VertexCounter (for IsSimplicialSurface)

▷ `VertexCounter(`*simpSurf*`)` (attribute)

Returns: a list of positive integers

Return the vertex counter of a simplicial surface. The vertex counter is defined as a list vertexSym, where vertexSym[i] counts the number of vertices that are incident to exactly i edges. If there are no vertices with i incident edges, this entry is unbounded.

2.3.10 EdgeCounter (for IsSimplicialSurface)

▷ `EdgeCounter(`*simpSurf*`)` (attribute)

Returns: a matrix of integers

Return the edge counter of a simplicial surface. The edge counter is a symmetric matrix M such that M[i,j] counts the number of edges such that the two vertices of the edge have edge-degrees i and j.

2.3.11 FaceAnomalyClasses (for IsSimplicialSurface)

▷ `FaceAnomalyClasses(`*simpSurf*`)` (attribute)

Returns: The face-anomaly-classes (as a list of sets)

Return the face-anomaly-classes of a simplicial surface.

Two faces are in the same face-anomaly-class if they contain the same vertices.

2.3.12 EdgeAnomalyClasses (for IsSimplicialSurface)

▷ `EdgeAnomalyClasses(`*simpSurf*`)` (attribute)

Returns: The edge-anomaly-classes (as a list of sets)

Return the edge-anomaly-classes of a simplicial surface (two edges are in the same edge-anomaly-class if they contain the same vertices).

2.3.13 IsAnomalyFree (for IsSimplicialSurface)

▷ `IsAnomalyFree(simpSurf)` (property)

Returns: true or false

Return whether the simplicial surface contains no anomalies (of faces or edges). This property is also known as vertex faithful. A simplicial surface is vertex faithful if all face and edge anomaly classes are trivial.

2.3.14 IncidenceGraph (for IsSimplicialSurface)

▷ `IncidenceGraph(simpSurf)` (attribute)

Returns: the coloured incidence graph

Return the coloured incidence graph of a simplicial surface.

- The vertex set of this graph consists of all vertices, edges and faces of the simplicial surface. All vertices, all edges and all faces are in individual colour classes.
- The edges are given by vertex-edge and edge-face pairs.

2.4 Functions for simplicial surfaces

2.4.1 SubsurfaceByFaces (for IsSimplicialSurface, IsSet)

▷ `SubsurfaceByFaces(simpSurf, faces)` (operation)

▷ `SubsurfaceByFacesNC(simpSurf, faces)` (operation)

Returns: a simplicial surface

Return the subsurface of a simplicial surface that is defined by the given set of faces.

The NC-version does not check if the given faces actually are faces of the simplicial surface.

2.4.2 SnippOffEars (for IsSimplicialSurface)

▷ `SnippOffEars(simpSurf)` (operation)

▷ `SnippOffEarsRecursive(simpSurf)` (operation)

Returns: a simplicial surface

Remove all ears of the simplicial surface and return the resulting surface. An ear is a face that has at most two vertices in common with all other faces.

The recursive-version applies this method recursively until the resulting simplicial surface has no more ears.

2.4.3 IsIsomorphic (for IsSimplicialSurface, IsSimplicialSurface)

▷ `IsIsomorphic(s1, s2)` (operation)

Returns: true or false

Check if two simplicial surfaces are isomorphic. This method only checks if they are isomorphic with respect to the incidence relation. It does not check if additional structure like a wild coloring is isomorphic (or even present).

2.4.4 AddVertexIntoEdge (for IsSimplicialSurface, IsPosInt)

▷ `AddVertexIntoEdge(simpSurf, edge)` (operation)

Returns: the modified simplicial surface

Add a vertex into an edge. This only works if there are exactly two faces adjacent to the edge.

2.5 Advanced properties of simplicial surfaces

Since the SimplicialSurface-objects in GAP can represent more general structures than just surfaces there is a property that checks whether a generic SimplicialSurface-objects represents an actual surface (the property IsActualSurface). To check whether a generic SimplicialSurface-object represents an actual surface we have to check the edges and the vertices.

- In an actual surface every edge is incident to at most two faces. This property is checked by IsEdgesLikeSurface.
- If each edge is incident to at most two faces, we can define face-edge-paths around each vertex. A face-edge-path around a vertex v is a list $(e_1, f_1, e_2, f_2, \ldots, e_n, f_n)$ or $(e_1, f_1, e_2, f_2, \ldots, e_n, f_n, e_{n+1})$ such that
  - all e_i are pairwise distinct edges incident to the vertex v
  - all f_i are pairwise distinct faces incident to the vertex v
  - if two elements are adjacent in a face-edge-path, they are incident in the simplicial surface
  - if the face-edge-path has even length (the first case), we require that e_1 and f_n are incident (this represents a closed path)
  - if the face-edge-path has odd length (the second case), we require that both e_1 and e_{n+1} are only incident to one face (this represents an open path) In general there may be many of those paths around a vertex (they partition the edges and faces incident to each vertex) but in an actual surface there is only one such path. The property IsVerticesLikeSurface checks this property.

2.5.1 IsVerticesLikeSurface (for IsSimplicialSurface)

▷ `IsVerticesLikeSurface(simpSurf)` (property)

Returns: `true` or `false`

Under the assumption that every edge is incident to at most two faces (which allows the definition of face-edge-paths) the property IsVerticesLikeSurface holds if there is only one face-edge-path (up to description) around each vertex.

2.5.2 IsEdgesLikeSurface (for IsSimplicialSurface)

▷ `IsEdgesLikeSurface(simpSurf)` (property)

Returns: `true` or `false`

The property IsEdgesLikeSurface holds if every edge of the simplicial surface is incident to at most two faces.

2.5.3 IsClosedSurface (for IsSimplicialSurface and IsEdgesLikeSurface)

▷ `IsClosedSurface(`*simpSurf*`)` (property)

Returns: `true` or `false`

The property IsActualSurface is true if both IsEdgesLikeSurface and IsVerticesLikeSurface are true. If we have a simplicial surface where every edge is incident to at most two faces, this method checks if the surface is closed. (A simplicial surface is closed if every edge is incident to exactly two faces.)

2.5.4 InnerEdges (for IsSimplicialSurface)

▷ `InnerEdges(`*simpSurf*`)` (attribute)

Returns: a set of edges

Return the set of all inner edges, that is edges with exactly two ajacent faces.

2.5.5 BoundaryEdges (for IsSimplicialSurface)

▷ `BoundaryEdges(`*simpSurf*`)` (attribute)

Returns: a set of edges

Return the set of all border edges, that is edges with only one adjacent face.

2.5.6 RamifiedEdges (for IsSimplicialSurface)

▷ `RamifiedEdges(`*simpSurf*`)` (attribute)

Returns: a set of edges

Return the set of all ramified edges, that is edges that have at least three adjacent faces.

2.5.7 FaceEdgePathsOfVertices (for IsSimplicialSurface and IsEdgesLikeSurface)

▷ `FaceEdgePathsOfVertices(`*simpSurf*`)` (attribute)
▷ `FaceEdgePathsOfVertex(`*simpSurf, vertex*`)` (operation)
▷ `FaceEdgePathsOfVertexNC(`*simpSurf, vertex*`)` (operation)

Returns: a list of lists of face-edge-paths

Return a list fep with the following conditions:

- for each vertex v the entry fep[v] contains a list of all face-edge-paths of this vertex. If there are several possible ways to describe a face-edge-path we choose the representative where the first face is minimal among all faces in the path. If this is not unique we choose the edge between the first and second face to be minimal among all remaining choices.
- every other entry of fep is not bounded.

If a vertex is given additionally, return only a list of face-edge-paths around this vertex. The NC-version does not check whether the given vertex actually is a vertex of the simplicial surface.

2.6 Local and global orientations

If we consider an embedding of a simplicial surface, each face gets mapped onto a polygon (usually a triangle). For each such polygon there are exactly two normal vectors that define the two "sides" of the polygon. Via the right-hand-rule from physics we can identify each normal vector with a cyclic permutation of the vertices in the polygon. For example, if a given face is incident to the vertices $\{1,2,3\}$, then the possible cyclic permutations are (1,2,3) and (1,3,2). If the polygon has more than three vertices then adjacent vertices in these permutations have to be connected by an edge.

The same construction can be used to derive a cyclic permutation of the incident edges. In addition it is sometimes useful to encode these permutations as lists, for example the permutation (1,2,3) corresponds to the list [1,2,3] (the first entry has to be the smallest one to make this choice of list unique).

A *local orientation* of a given face can now be described in four equivalent ways:

- ByVerticesAsPerm

  This is a cyclic permutation p of all vertices incident to the face such that v and p(v) are incident to an edge of the face (for each vertex v in the face).

- ByVerticesAsList

  This is a list of all vertices indicent to the face that encodes the permutation of *ByVerticesAsPerm*. The first entry is the smallest vertex among those incident to the face.

- ByEdgesAsPerm

  This is a cyclic permutation p of all edges incident to the face such that e and p(e) are incident to an edge of the face (for each edge e in the face).

- ByEdgesAsList

  This is a list of all edges indicent to the face that encodes the permutation of *ByEdgesAsPerm*. The first entry is the smallest edge among those incident to the face.

A standard local orientation is used to distinguish the different sides for each face. There is no necessary connection between local orientations of different faces. It is possible to give the sides different names. If no special measures are taken, the side that is distinguished by the standard local orientation is known as f, whereas the other side is known as -f.

If the simplicial surface is orientable (which is only well-defined if the property IsEdgesLikeSurface is true), we can assign a global orientation. More specifically there are 2^c different global orientations, where c denotes the number of path-connected components of the simplicial surface (two faces are path-connected if there exists a face-edge-path between them that is not necessarily incident to only one vertex).

To return a unique global orientation we pick the smallest face in each path-connected component and assign the standard local orientation to this face.

To describe the distinguished global orientation we have the same options as in the local case.

2.6.1 LocalOrientationByVerticesAsPerm (for IsSimplicialSurface)

▷ `LocalOrientationByVerticesAsPerm(`*`simpSurf`*`)` (attribute)
▷ `LocalOrientation(`*`simpSurf`*`)` (operation)

Returns: a list of permutations

Return a list localOr with the following properties:

- For each face f the entry localOr[f] consists of a cycle with all vertices that are incident to f. Adjacent vertices have an edge of the face in common.
- All other positions are not bounded.

This is the default interpretation for the method LocalOrientation.

2.6.2 LocalOrientationByVerticesAsList (for IsSimplicialSurface)

▷ `LocalOrientationByVerticesAsList(`*simpSurf*`)` (attribute)

Returns: a list of lists

Return a list localOr with the following properties:

- For each face f the entry localOr[f] consists of a list with all vertices that are incident to f. Adjacent vertices have an edge of the face in common.
- All other positions are not bounded.

2.6.3 LocalOrientationByEdgesAsPerm (for IsSimplicialSurface)

▷ `LocalOrientationByEdgesAsPerm(`*simpSurf*`)` (attribute)

Returns: a list of permutations

Return a list localOr with the following properties:

- For each face f the entry localOr[f] consists of a cycle with all edges that are incident to f. Adjacent edges have an vertex of the face in common.
- All other positions are not bounded.

2.6.4 LocalOrientationByEdgesAsList (for IsSimplicialSurface)

▷ `LocalOrientationByEdgesAsList(`*simpSurf*`)` (attribute)

Returns: a list of permutations

Return a list localOr with the following properties:

- For each face f the entry localOr[f] consists of a list with all edges that are incident to f. Adjacent edges have an vertex of the face in common.
- All other positions are not bounded.

2.6.5 NamesOfFaces (for IsSimplicialSurface)

▷ `NamesOfFaces(`*simpSurf*`)` (attribute)
▷ `NamesOfFace(`*simpSurf*`, `*face*`)` (operation)
▷ `NamesOfFaceNC(`*simpSurf*`, `*face*`)` (operation)

Returns: a list of lists of integers

Return alist names with the following properties:

- For each face f the entry names[f] is a list of two integers. The first integer is the name of the upper face-side, the second one is the name of the lower face-side (with respect to the local orientation).

- all other entries are unbounded.

If a face is given in addition, the corresponding entry of this list is returned. The NC-version does not throw an error if a non-face is given.

2.6.6 FaceByName (for IsSimplicialSurface, IsInt)

▷ `FaceByName(simpSurf, name)` (operation)

Returns: a positive integer

Return the face of the simplicial surface that has the given name as the name of one of its sides.

2.6.7 IsFaceNamesDefault (for IsSimplicialSurface)

▷ `IsFaceNamesDefault(simpSurf)` (property)

Returns: `true` or `false`

Return whether the naming scheme for the faces is the default one, meaning that the upper side of a face f is called f (a positive integer) and the lower side -f (a negative integer).

2.6.8 IsOrientable (for IsSimplicialSurface and IsEdgesLikeSurface)

▷ `IsOrientable(simpSurf)` (property)

Returns: `true` or `false`

If we have a simplicial surface where every edge is incident to at most two faces, this method checks if the surface is orientable. (A simplicial surface is orientable if we can assign a side for each face such that for every two adjacent sides either both or none are assigned.

2.6.9 GlobalOrientationByVerticesAsPerm (for IsSimplicialSurface and IsEdgesLikeSurface)

▷ `GlobalOrientationByVerticesAsPerm(simpSurf)` (attribute)

▷ `GlobalOrientation(simpSurf)` (operation)

Returns: a list of permutations or fail

Return the distinguished global orientation if the simplicial surface is orientable. Warning: The returned orientation depends on the chosen local orientation of the simplicial surface. This might not be equal even if the incidence structure is.

The orientation is returned as a list globalOr. For each face f the entry globalOr[f] contains a cycle of the vertices that are incident in the face f. The order of this cycle corresponds to the orientation of the face f with respect to the global orientability.

This method returns fail if the given surface is not orientable.

2.6.10 GlobalOrientationByVerticesAsList (for IsSimplicialSurface and IsEdgesLikeSurface)

▷ `GlobalOrientationByVerticesAsList(simpSurf)` (attribute)

Returns: a list of lists or fail

Return the distinguished global orientation if the simplicial surface is orientable. Warning: The returned orientation depends on the chosen local orientation of the simplicial surface. This might not be equal even if the incidence structure is.

The orientation is returned as a list globalOr. For each face f the entry globalOr[f] contains a list of the vertices that are incident in the face f. The order of this list corresponds to the orientation of the face f with respect to the global orientability.

This method returns fail if the given surface is not orientable.

2.6.11 GlobalOrientationByEdgesAsPerm (for IsSimplicialSurface and IsEdgesLikeSurface)

▷ `GlobalOrientationByEdgesAsPerm(`*simpSurf*`)` (attribute)

Returns: a list of permutations or fail

Return the distinguished global orientation if the simplicial surface is orientable. Warning: The returned orientation depends on the chosen local orientation of the simplicial surface. This might not be equal even if the incidence structure is.

The orientation is returned as a list globalOr. For each face f the entry globalOr[f] contains a cycle of the edges that are incident in the face f. The order of this cycle corresponds to the orientation of the face f with respect to the global orientability.

This method returns fail if the given surface is not orientable.

2.6.12 GlobalOrientationByEdgesAsList (for IsSimplicialSurface and IsEdgesLikeSurface)

▷ `GlobalOrientationByEdgesAsList(`*simpSurf*`)` (attribute)

Returns: a list of lists or fail

Return the distinguished global orientation if the simplicial surface is orientable. Warning: The returned orientation depends on the chosen local orientation of the simplicial surface. This might not be equal even if the incidence structure is.

The orientation is returned as a list globalOr. For each face f the entry globalOr[f] contains a list of the edges that are incident in the face f. The order of this list corresponds to the orientation of the face f with respect to the global orientability.

This method returns fail if the given surface is not orientable.

2.7 Technical functions (for development)

This section contains methods that concern the internal structure of simplicial surfaces. You only have to read this section if you want to understand the underlying implementation better or if you want to develop code that is derived from this. There are three unique features in the implementation of simplicial surfaces that especially concern the definition of specializes simplicial surfaces:

- The use of a method selection graph
- A general methods to help defining specialized classes
- A guide for easier initialization

Since the method selection graph is the most salient feature we will cover it first. It derives from a simple observation: If you know either VerticesOfEdges or EdgesOfVertices, you can calculate the other. If you additionally know either of EdgesOfFaces or FacesOfEdges, you can calculate all six of these attributes. This could have been implemented by a lot of specialized methods but the number of

these methods rised exponentially with the number of attributes that are connected. Instead we only implement methods for the "difficult" parts (where work has to be done) and delegate the "easy" parts (if we can calculate B from A and C from B, we can also calculate C from A) into a method selection graph. If an attribute should be part of the method selection graph (which it only should if you can calculate information inside the method selection graph from this attribute) you have to make two modifications:

- There has to be a method to calculate this attribute by the method selection graph, like
- For each "difficult" method there has to be a call that adds this possibility into the method selection graph, like

Secondly we guarantee unique methods for specialization. To consider a specific example, imagine we want to give certain simplicial surfaces an edge colouring. If one simplicial surface may have different edge colourings we can't implement this as an attribute of the simplicial surface. Instead we define a new type for this situation (as a subtype of SimplicialSurfaceType). The disadvantage of this procedure is that is becomes harder to take a simplicial surface as input and add an edge colouring (type changes are frowned upon in GAP). For this reason we offer a special method that does just that - it copies many attributes of the simplicial surface into an object of the new type.

2.7.1 ObjectifySimplicialSurface (for IsType,IsRecord,IsSimplicialSurface)

▷ `ObjectifySimplicialSurface(`*type, record, simpSurf*`)` (operation)

Returns: an object of type type

This function calls and afterwards copies all attributes and properties of the simplicial surface modelSurf that are declared in this section to the the new object. This method has to be overwritten for a specialization of this class. WARNING: The type can't be checked! Only types that are derived from IsSimplicialSurface can be used with impunity!

Finally we consider the constructors. Simplicial surfaces are defined by an incidence structure *and* a local orientation (with face names). However, only the incidence structure has to be given - the local orientation can be derived (in the sense that there are several possibilites and one of them will be picked). To facilitate this procedure we offer a method which can be called *after* the incidence structure is defined.

2.7.2 DeriveLocalOrientationAndFaceNamesFromIncidenceGeometry (for IsSimplicialSurface)

▷ `DeriveLocalOrientationAndFaceNamesFromIncidenceGeometry(`*simpSurf*`)` (operation)

▷ `DeriveLocalOrientationAndFaceNamesFromIncidenceGeometryNC(`*arg*`)` (operation)

Returns: nothing

This is a method which should only be used in code development. It should not be called by a normal user as it presupposes knowledge of the internal attribute storing system. A simplicial surface consists of two separate sets of attributes: One set of attributes to save the incidence geometry (Vertices, Edges, Faces, EdgesOfFaces, etc.), the other to save the local orientation of the faces (LocalOrientationOfVerticesAsPerm, NamesOfFaces, etc.). While the second set of attributes may be crucial for some applications (like folding), it is easy to ignore it in other applications. This is usually managed by a judicious constructor call that will handle the necessary overhead without burdening the user. If - for whatever reason - no constructor should be called (for example for a subcategory of

IsSimplicialSurface) this method can be used to initialize all necessary attributes of the second set. This method will throw an error if some of these attributes are already set. It will only check the attributes

- LocalOrientationByVerticesAsPerm
- LocalOrientationByVerticesAsList
- LocalOrientationByEdgesAsPerm
- LocalOrientationByEdgesAsList
- IsFaceNamesDefault
- NamesOfFaces

If other attributes interfere with these, they will not be checked! For this reason this method should only be called if one knows exactly which attributes are already set. The NC-version doesn't check whether attributes are set (it is therefore even more dangerous to use).

2.7.3 PrintStringAttributeOfSimplicialSurface (for IsSimplicialSurface)

▷ `PrintStringAttributeOfSimplicialSurface(`*simpSurf*`)` (attribute)

Returns: a string

Return the string that is printed by the PrintObj-method. This method is used since it may be expensive to compute the string.

Chapter 3

Wild Simplicial Surfaces

3.1 Constructors for wild simplicial surfaces

3.1.1 WildSimplicialSurfaceByDownwardIncidenceAndEdgeColouring (for IsSet, IsSet, IsSet, IsList, IsList, IsList)

▷ `WildSimplicialSurfaceByDownwardIncidenceAndEdgeColouring(`*`vertices, edges, faces, verticesOfEdges, edgesOfFaces`*`)` (operation)

Returns: a wild simplicial surface

This is a constructor for a wild simplicial surface based on the following information: - vertices A set of vertices (a positive integer n may represent [1..n]) - edges A set of edges (a positive integer n may represent [1..n]) - faces A set of faces (a positive integer n may represent [1..n]) - verticesOfEdges A list of sets, where a set of two vertices is given for each edge - edgesOfFaces A list of sets, where a set of three edges is given for each face - coloursOfEdges A list that contains the numbers 1,2,3. For each edge the colour of this edge is given. There have to be an equal amount of edges for each colour and the edges of each face have to have different colours. The NC-version doesn't check whether the given information is consistent. coloursOfEdges

3.1.2 WildSimplicialSurfaceByDownwardIncidenceAndEdgeColouringNC (for IsSet, IsSet, IsSet, IsList, IsList, IsList)

▷ `WildSimplicialSurfaceByDownwardIncidenceAndEdgeColouringNC(`*`arg1, arg2, arg3, arg4, arg5, arg6`*`)` (operation)

Returns:

3.1.3 WildSimplicialSurfaceByDownwardIncidenceAndGenerators (for IsSet, IsSet, IsSet, IsList, IsList, IsList)

▷ `WildSimplicialSurfaceByDownwardIncidenceAndGenerators(`*`arg1, arg2, arg3, arg4, arg5, arg6`*`)` (operation)

Returns:

This is a constructor for a wild simplicial surface based on the following information: - vertices A set of vertices (a positive integer n may represent [1..n]) - edges A set of edges (a positive integer n may represent [1..n]) - faces A set of faces (a positive integer n may represent [1..n]) - verticesOfEdges A list of sets, where a set of two vertices is given for each edge - edgesOfFaces A list of sets, where

a set of three edges is given for each face - generators A list of three involutions whose cycles define the edge colouring. If this is not unique (that is, if there is are two edges that are incident to the same two faces, then an error is thrown). The NC-version doesn't check whether the given information is consistent.

3.1.4 WildSimplicialSurfaceByDownwardIncidenceAndGeneratorsNC (for IsSet, IsSet, IsSet, IsList, IsList, IsList)

▷ `WildSimplicialSurfaceByDownwardIncidenceAndGeneratorsNC(`*arg1, arg2, arg3, arg4, arg5, arg6*`)` (operation)

Returns:

3.1.5 WildSimplicialSurfaceExtensionByEdgeColouring (for IsSimplicialSurface and IsActualSurface and IsTriangleSurface, IsList)

▷ `WildSimplicialSurfaceExtensionByEdgeColouring(`*arg1, arg2*`)` (operation)

Returns:

This is a constructor for a wild simplicial surface based on the following information: - surface A simplicial surface which consists only of triangles and is an actual surface. - coloursOfEdges A list that contains the numbers 1,2,3. For each edge the colour of this edge is given. There have to be an equal amount of edges for each colour and the edges of each face have to have different colours. The NC-version doesn't check whether the given information is consistent.

3.1.6 WildSimplicialSurfaceExtensionByEdgeColouringNC (for IsSimplicialSurface and IsActualSurface and IsTriangleSurface, IsList)

▷ `WildSimplicialSurfaceExtensionByEdgeColouringNC(`*arg1, arg2*`)` (operation)

Returns:

3.1.7 WildSimplicialSurfaceExtensionByGenerators (for IsSimplicialSurface and IsActualSurface and IsTriangleSurface, IsList)

▷ `WildSimplicialSurfaceExtensionByGenerators(`*arg1, arg2*`)` (operation)

Returns:

This is a constructor for a wild simplicial surface based on the following information: - surface A simplicial surface which consists only of triangles and is an actual surface. - generators A list of three involutions whose cycles define the edge colouring. If this is not unique (that is, if there is are two edges that are incident to the same two faces, then an error is thrown). The NC-version doesn't check whether the given information is consistent.

3.1.8 WildSimplicialSurfaceExtensionByGeneratorsNC (for IsSimplicialSurface and IsActualSurface and IsTriangleSurface, IsList)

▷ `WildSimplicialSurfaceExtensionByGeneratorsNC(`*arg1, arg2*`)` (operation)

Returns:

3.1.9 WildSimplicialSurfaceByFaceEdgesPathsAndEdgeColouring (for IsSet, IsSet, IsSet, IsList, IsList)

▷ `WildSimplicialSurfaceByFaceEdgesPathsAndEdgeColouring(`*arg1, arg2, arg3, arg4, arg5*`)` (operation)

Returns:

This is a constructor for a wild simplicial surface based on the following information: - vertices A set of vertices (a positive integer n may represent [1..n]) - edges A set of edges (a positive integer n may represent [1..n]) - faces A set of faces (a positive integer n may represent [1..n]) - faceEdgePaths A list of the face-edge-paths for each vertex. - coloursOfEdges A list that contains the numbers 1,2,3. For each edge the colour of this edge is given. There have to be an equal amount of edges for each colour and the edges of each face have to have different colours. The NC-version doesn't check whether the given information is consistent.

3.1.10 WildSimplicialSurfaceByFaceEdgesPathsAndEdgeColouringNC (for IsSet, IsSet, IsSet, IsList, IsList)

▷ `WildSimplicialSurfaceByFaceEdgesPathsAndEdgeColouringNC(`*arg1, arg2, arg3, arg4, arg5*`)` (operation)

Returns:

3.1.11 WildSimplicialSurfaceByColouredFaceEdgePaths (for IsSet, IsSet, IsList)

▷ `WildSimplicialSurfaceByColouredFaceEdgePaths(`*arg1, arg2, arg3*`)` (operation)

Returns:

This is a constructor for a wild simplicial surface based on the following information (the edges will be constructed internally): - vertices A set of vertices (a positive integer n may represent [1..n]) - faces A set of faces (a positive integer n may represent [1..n]) - colfaceEdgePaths A list of the coloured face-edge-paths for each vertex. The NC-version doesn't check whether the given information is consistent.

3.1.12 WildSimplicialSurfaceByColouredFaceEdgePathsNC (for IsSet, IsSet, IsList)

▷ `WildSimplicialSurfaceByColouredFaceEdgePathsNC(`*arg1, arg2, arg3*`)` (operation)

Returns:

3.2 Attributes and properties of wild coloured simplicial surfaces

3.2.1 Generators (for IsWildSimplicialSurface)

▷ `Generators(a, `*wild, simplicial, surface*`)` (attribute)

Returns: a dense list of permutations

Returns the generators of the wild simplicial surface in a list.

3.2.2 GroupOfWildSimplicialSurface (for IsWildSimplicialSurface)

▷ `GroupOfWildSimplicialSurface(a, `*wild, simplicial, surface*`)` (attribute)

Returns: a group

Return the group that is generated by the generators of the wild simplicial surface.

3.2.3 VertexGroup (for IsWildSimplicialSurface)

▷ `VertexGroup(arg)` (attribute)

Returns: finitely presented group.

Given a wild coloured simplicial surface simpsurf, this function determines the vertex group of the simplicial surface. The vertex group of the simplicial surface simpsurf is defined to be F_3/R, where F_3 is the free group on three generators and R is the set of relations given by the vertex defining paths of the inner vertices.

3.2.4 ColoursOfEdges (for IsWildSimplicialSurface)

▷ `ColoursOfEdges(a, wild, simplicial, surface)` (attribute)

Returns: a list

Return the edge colors. We return a list which has unbounded entries for all edges (and only those). At the position 'edge' the colour of this edge is saved - that is, to which generator this edge belongs.

3.2.5 ColourOfEdge (for IsWildSimplicialSurface, IsPosInt)

▷ `ColourOfEdge(arg1, arg2)` (operation)

Returns:

3.2.6 ColourOfEdgeNC (for IsWildSimplicialSurface, IsPosInt)

▷ `ColourOfEdgeNC(arg1, arg2)` (operation)

Returns:

3.2.7 ColouredEdgesOfFaces (for IsWildSimplicialSurface)

▷ `ColouredEdgesOfFaces(a, wild, simplicial, surface)` (attribute)

Returns: a list

Return a list which is indexed by the faces. For each face we get a list of the incident edges such that the first entry corresponds to the edge of the first colour, the second entry to the edge of the second colour and the third entry to the edge of the third colour.

3.2.8 ColouredEdgeOfFace (for IsWildSimplicialSurface, IsPosInt, IsPosInt)

▷ `ColouredEdgeOfFace(arg1, arg2, arg3)` (operation)

Returns:

3.2.9 ColouredEdgeOfFaceNC (for IsWildSimplicialSurface, IsPosInt, IsPosInt)

▷ `ColouredEdgeOfFaceNC(arg1, arg2, arg3)` (operation)

Returns:

3.2.10 ColouredFaceEdgePathsOfVertices (for IsWildSimplicialSurface)

▷ ColouredFaceEdgePathsOfVertices(a, *wild, simplicial, surface*) (attribute)

Returns: a list

A coloured face edge path of an inner vertex v of a wild simplicial surface is a list $[c_1, f_1, c_2, f_2, ..., c_n, f_n]$, where f1,...,fn are the faces incident to v and c1,..,cn are colours such that the edges of face fi with ci and c(i+1) are those incident to v and c(n+1)=c1. A coloured face edge path of a boundary vertex v of a wild simplicial surface is a list [c1,f1,c2,f2,...,cn,fn, c(n+1)], where f1,...,fn are the faces incident to v and c1,..,c(n+1) are colours such that the edges of face fi with ci and c(i+1) are those incident to v and the edges of f1 and fn with colours c1 and c(n+1), respectively, are boundary edges.

3.2.11 ColouredFaceEdgePathOfVertex (for IsWildSimplicialSurface,IsPosInt)

▷ ColouredFaceEdgePathOfVertex(*arg1, arg2*) (operation)

Returns:

3.2.12 ColouredFaceEdgePathOfVertexNC (for IsWildSimplicialSurface,IsPosInt)

▷ ColouredFaceEdgePathOfVertexNC(*arg1, arg2*) (operation)

Returns:

3.2.13 EdgesOfColours (for IsWildSimplicialSurface)

▷ EdgesOfColours(a, *wild, simplicial, surface*) (attribute)

Returns: a list of sets of edges

Return a list of three sets, where the first set consists of all edges of the first colour, and so on.

3.2.14 EdgesOfColour (for IsWildSimplicialSurface, IsPosInt)

▷ EdgesOfColour(*arg1, arg2*) (operation)

Returns:

3.2.15 EdgesOfColourNC (for IsWildSimplicialSurface, IsPosInt)

▷ EdgesOfColourNC(*arg1, arg2*) (operation)

Returns:

3.2.16 MRTypeOfEdges (for IsWildSimplicialSurface)

▷ MRTypeOfEdges(*simpsurf*, a, *simplicial, surface, object, as, created*) (attribute)

Returns: a list of three lists, each of which contains the entries 0, 1 or 2.

Given a wild coloured simplicial surface simpsurf, this function determines the mr-type of each of the edges of simpsurf. The mr-type of an edge of simpsurf is either "m" (for mirror) or "r" (for rotation). It is defined as followed. Suppose the edge e is incident to the vertices v_1 and v_2 and to the two faces F and F'. Let x and y be the edges of incident incident to F and F' and to the same vertex v_1, say. Then e is of type m if both x and y have the same colour, and e is of type r if x and y are different. As we assume the surface to be wild coloured, this means that the colours of the other edges incident to e and both faces F and F' are then also determined. As the # $'$ edges of the simplicial surface are

pairs of points, the mr-type of the simplicial surface simpsurf can be encoded as a list of length 3. Each of the entries is in turn a list encoding the mr-type of all edges of a certain colour. Suppose that mrtype[1] is the list encoding the mr-type of the red edges. Then mrtype[1][i] = 0 if the mr-type of the red edge incident to the vertex i is unknown, mrtype[1][i] = 1 if the mr-type of the red edge incident to the vertex i is "m", and mrtype[1][i] = 2 if the mr-type of the red edge incident to the vertex i is "r". by WildSimplicialSurface

3.2.17 MRTypeOfEdgesAsNumbers (for IsWildSimplicialSurface)

▷ `MRTypeOfEdgesAsNumbers(arg)` (attribute)

Returns:

3.2.18 IsSurfaceWithStructure (for IsWildSimplicialSurface)

▷ `IsSurfaceWithStructure(wildSurf)` (property)

Returns: true or false

If the mr-types of all edges with the same colour are identical, we call this a structure of the surface. This property checks whether this is the case.

3.2.19 MRTypeOfSurfaceWithStructure (for IsWildSimplicialSurface and IsSurfaceWithStructure)

▷ `MRTypeOfSurfaceWithStructure(wildSurf)` (attribute)

Returns: a list

Return the MR-type of a surface with structure in the form of a list [type of first colour, type of second colour, type of third colour].

3.2.20 MRTypeOfSurfaceWithStructureAsNumbers (for IsWildSimplicialSurface and IsSurfaceWithStructure)

▷ `MRTypeOfSurfaceWithStructureAsNumbers(arg)` (attribute)

Returns:

3.3 Functions for wild coloured simplicial surfaces

3.3.1 EdgeByFacesAndColour (for IsWildSimplicialSurface, IsSet, IsPosInt)

▷ `EdgeByFacesAndColour(wildSimplicialSurface, setOfFaces, colour)` (operation)

Returns: an edge

Given a set of faces and a colour, return the edge such that the set of faces is the set of faces incident to the edge and that the edge has the given colour. The NC-version doesn't check if the given colour is valid.

3.3.2 EdgeByFacesAndColourNC (for IsWildSimplicialSurface, IsSet, IsPosInt)

▷ `EdgeByFacesAndColourNC(arg1, arg2, arg3)` (operation)

Returns:

3.3.3 DoubleCover (for IsWildSimplicialSurface)

▷ `DoubleCover(`*`wildSimplicialSurface`*`)` (operation)

Returns: the double cover

Return the double cover of the wild simplicial surface, keeping the MR-type fixed.

3.3.4 SixFoldCover (for IsSimplicialSurface, IsList)

▷ `SixFoldCover(`*`surface, mrtype, bool`*`)` (operation)

Returns: a wild coloured simplicial surface if bool is true or not given, the mrtype specifies the behaviour of the σ_i, if bool is false, the mrtype specifies the behaviour of the edges.

The function SixFoldCover takes as input a generic description of a simplicial surface. The six fold cover of a simplicial surface is the following surface. If f is a face of the original face with edge numbers e_a, e_b and e_c, then the face is covered by the six faces of the form (f, e1, e2, e3), for which {e1, e2, e3} = {e_a, e_b, e_c}. See Proposition 3.XX in the paper. If the optional argument mrtype is given, it has to be a list of length 3 and each entry has to be 1, or 2. In this case the six fold cover will treat the position i for $i \in \{1,2,3\}$ of the three edges around a faces either as a reflection (mirror), if the entry in position i of mrtype is 1, or as a rotation, if the entry in position i is 2. That is, the cover surface is generated by three transpositions σ_i for $i = 1,2,3$. For $i = 1$, suppose f and f' are faces of the surface surf such that the edges of f are e_1, e_2 and e_3 and the edges of f' are e_1, e_a, e_b are the edges e_1, e_2 and e_a intersect in a common vertex and the edges e_1, e_3 and e_b intersect in a common vertex. For $i = 1$ and mrtype of position 1 being mirror (i.e. 1), then

$$\sigma_1(f, e_1, e_2, e_3) = (f', e_1, e_a, e_b),$$

whereas if the mrtype of position 1 is a reflection (i.e. 2), then

$$\sigma_1(f, e_1, e_2, e_3) = (f', e_1, e_b, e_a).$$

The definition of σ_2 and σ_3 are analogous, with e_2, respectively e_3 taking the role of the common edge e_1. If the optional argument mredges is given, and mredges is a list of length equal to the number of edges of the surface *surf* and an entry for an edge e is either 1 or 2. If the entry is 1 then the six fold cover will treat the edge as a reflection (mirror) and if the entry is 2 then the edge is treated as a rotation. The six fold cover is always a wild colourable simplicial surface.

3.3.5 ImageWildSimplicialSurface (for IsWildSimplicialSurface, IsPerm)

▷ `ImageWildSimplicialSurface(`*`surface, perm`*`)` (operation)

Returns: The new wild simplicial surface

Compute the action of a permutation on the faces of a wild simplicial surface. The permutation acts on the faces, while the names of vertices and edges may change.

Index

AddVertexIntoEdge
for IsSimplicialSurface, IsPosInt, 14

BoundaryEdges
for IsSimplicialSurface, 15

ColouredEdgeOfFace
for IsWildSimplicialSurface, IsPosInt, IsPosInt, 25
ColouredEdgeOfFaceNC
for IsWildSimplicialSurface, IsPosInt, IsPosInt, 25
ColouredEdgesOfFaces
for IsWildSimplicialSurface, 25
ColouredFaceEdgePathOfVertex
for IsWildSimplicialSurface,IsPosInt, 26
ColouredFaceEdgePathOfVertexNC
for IsWildSimplicialSurface,IsPosInt, 26
ColouredFaceEdgePathsOfVertices
for IsWildSimplicialSurface, 26
ColourOfEdge
for IsWildSimplicialSurface, IsPosInt, 25
ColourOfEdgeNC
for IsWildSimplicialSurface, IsPosInt, 25
ColoursOfEdges
for IsWildSimplicialSurface, 25
ConnectedComponentOfFace
for IsSimplicialSurface, IsPosInt, 11
ConnectedComponentOfFaceNC
for IsSimplicialSurface, IsPosInt, 11
ConnectedComponents
for IsSimplicialSurface, 11
ConnectedComponentsAttributeOfSimplicialSurface
for IsSimplicialSurface, 11
Cube
for , 5

DeriveLocalOrientationAndFaceNamesFromIncidenceGeometry
for IsSimplicialSurface, 20
DeriveLocalOrientationAndFaceNamesFromIncidenceGeometryNC
for IsSimplicialSurface, 20
Dodecahedron
for , 5
DoubleCover
for IsWildSimplicialSurface, 28

EdgeAnomalyClasses
for IsSimplicialSurface, 12
EdgeByFacesAndColour
for IsWildSimplicialSurface, IsSet, IsPosInt, 27
EdgeByFacesAndColourNC
for IsWildSimplicialSurface, IsSet, IsPosInt, 27
EdgeCounter
for IsSimplicialSurface, 12
EdgeInFaceByVertices
for IsSimplicialSurface, IsPosInt, IsList, 10
Edges
for IsSimplicialSurface, 8
EdgesOfColour
for IsWildSimplicialSurface, IsPosInt, 26
EdgesOfColourNC
for IsWildSimplicialSurface, IsPosInt, 26
EdgesOfColours
for IsWildSimplicialSurface, 26
EdgesOfFaces
for IsSimplicialSurface, 10
EdgesOfVertices
for IsSimplicialSurface, 9
EulerCharacteristic
for IsSimplicialSurface, 10

FaceAnomalyClasses
for IsSimplicialSurface, 12
FaceByName

for IsSimplicialSurface, IsInt, 18
FaceEdgePathsOfVertex
for IsSimplicialSurface and IsEdgesLikeSurface, IsPosInt, 15
FaceEdgePathsOfVertexNC
for IsSimplicialSurface and IsEdgesLikeSurface, IsPosInt, 15
FaceEdgePathsOfVertices
for IsSimplicialSurface and IsEdgesLikeSurface, 15
Faces
for IsSimplicialSurface, 9
FacesOfEdges
for IsSimplicialSurface, 10
FacesOfVertices
for IsSimplicialSurface, 9

Generators
for IsWildSimplicialSurface, 24
GlobalOrientation
for IsSimplicialSurface and IsEdgesLikeSurface, 18
GlobalOrientationByEdgesAsList
for IsSimplicialSurface and IsEdgesLikeSurface, 19
GlobalOrientationByEdgesAsPerm
for IsSimplicialSurface and IsEdgesLikeSurface, 19
GlobalOrientationByVerticesAsList
for IsSimplicialSurface and IsEdgesLikeSurface, 18
GlobalOrientationByVerticesAsPerm
for IsSimplicialSurface and IsEdgesLikeSurface, 18
GroupOfWildSimplicialSurface
for IsWildSimplicialSurface, 24

Icosahedron
for , 5
ImageWildSimplicialSurface
for IsWildSimplicialSurface, IsPerm, 28
IncidenceGraph
for IsSimplicialSurface, 13
InnerEdges
for IsSimplicialSurface, 15
IsAnomalyFree
for IsSimplicialSurface, 13
IsClosedSurface
for IsSimplicialSurface and IsEdgesLikeSurface, 15
IsConnected
for IsSimplicialSurface, 11
IsEdgesLikeSurface
for IsSimplicialSurface, 14
IsFaceNamesDefault
for IsSimplicialSurface, 18
IsIsomorphic
for IsSimplicialSurface, IsSimplicialSurface, 13
IsOrientable
for IsSimplicialSurface and IsEdgesLikeSurface, 18
IsPathConnected
for IsSimplicialSurface, 11
IsSurfaceWithStructure
for IsWildSimplicialSurface, 27
IsTriangleSurface
for IsSimplicialSurface, 11
IsVerticesLikeSurface
for IsSimplicialSurface, 14

Janushead
for , 5

LocalOrientation
for IsSimplicialSurface, 16
LocalOrientationByEdgesAsList
for IsSimplicialSurface, 17
LocalOrientationByEdgesAsPerm
for IsSimplicialSurface, 17
LocalOrientationByVerticesAsList
for IsSimplicialSurface, 17
LocalOrientationByVerticesAsPerm
for IsSimplicialSurface, 16

MRTypeOfEdges
for IsWildSimplicialSurface, 26
MRTypeOfEdgesAsNumbers
for IsWildSimplicialSurface, 27
MRTypeOfSurfaceWithStructure
for IsWildSimplicialSurface and IsSurfaceWithStructure, 27
MRTypeOfSurfaceWithStructureAsNumbers
for IsWildSimplicialSurface and IsSurfaceWithStructure, 27

NamesOfFace
for IsSimplicialSurface, IsPosInt, 17
NamesOfFaceNC
for IsSimplicialSurface, IsPosInt, 17
NamesOfFaces
for IsSimplicialSurface, 17
NrOfEdges
for IsSimplicialSurface, 9
NrOfFaces
for IsSimplicialSurface, 9
NrOfVertices
for IsSimplicialSurface, 9

ObjectifySimplicialSurface
for IsType,IsRecord,IsSimplicialSurface, 20
Octahedron
for , 5

PathConnectedComponentOfFace
for IsSimplicialSurface, IsPosInt, 11
PathConnectedComponentOfFaceNC
for IsSimplicialSurface, IsPosInt, 11
PathConnectedComponents
for IsSimplicialSurface, 11
PrintStringAttributeOfSimplicial-
Surface
for IsSimplicialSurface, 21

RamifiedEdges
for IsSimplicialSurface, 15

SimplicialSurfaceByDownwardIncidence
for IsSet, IsSet, IsSet, IsList, IsList, 6
SimplicialSurfaceByDownwardIncidenceNC
for IsSet, IsSet, IsSet, IsList, IsList, 6
SimplicialSurfaceByDownwardIncidence-
WithOrientation
for IsSet, IsSet, IsSet, IsList, IsList, 7
SimplicialSurfaceByDownwardIncidence-
WithOrientationNC
for IsSet, IsSet, IsSet, IsList, IsList, 7
SimplicialSurfaceByUpwardIncidence
for IsSet, IsSet, IsSet, IsList, IsList, 5
SimplicialSurfaceByUpwardIncidenceNC
for IsSet, IsSet, IsSet, IsList, IsList, 5
SimplicialSurfaceByVerticesInFaces
for IsSet, IsSet, IsList, 8
SimplicialSurfaceByVerticesInFacesNC
for IsSet, IsSet, IsList, 8
SixFoldCover
for IsSimplicialSurface, IsList, 28
SnippOffEars
for IsSimplicialSurface, 13
SnippOffEarsRecursive
for IsSimplicialSurface, 13
SortedDegrees
for IsSimplicialSurface, 12
SubsurfaceByFaces
for IsSimplicialSurface, IsSet, 13
SubsurfaceByFacesNC
for IsSimplicialSurface, IsSet, 13

Tetrahedron
for , 5

UnsortedDegrees
for IsSimplicialSurface, 12

VertexCounter
for IsSimplicialSurface, 12
VertexGroup
for IsWildSimplicialSurface, 25
Vertices
for IsSimplicialSurface, 8
VerticesAttributeOfSimplicialSurface
for IsSimplicialSurface, 8
VerticesOfEdges
for IsSimplicialSurface, 9
VerticesOfFaces
for IsSimplicialSurface, 10

WildSimplicialSurfaceByColouredFace-
EdgePaths
for IsSet, IsSet, IsList, 24
WildSimplicialSurfaceByColouredFace-
EdgePathsNC
for IsSet, IsSet, IsList, 24
WildSimplicialSurfaceByDownward-
IncidenceAndEdgeColouring
for IsSet, IsSet, IsSet, IsList, IsList, IsList, 22
WildSimplicialSurfaceByDownward-
IncidenceAndEdgeColouringNC
for IsSet, IsSet, IsSet, IsList, IsList, IsList, 22

WildSimplicialSurfaceByDownward-
IncidenceAndGenerators
for IsSet, IsSet, IsSet, IsList, IsList, IsList, 22
WildSimplicialSurfaceByDownward-
IncidenceAndGeneratorsNC
for IsSet, IsSet, IsSet, IsList, IsList, IsList, 23
WildSimplicialSurfaceByFaceEdgesPaths-
AndEdgeColouring
for IsSet, IsSet, IsSet, IsList, IsList, 24
WildSimplicialSurfaceByFaceEdgesPaths-
AndEdgeColouringNC
for IsSet, IsSet, IsSet, IsList, IsList, 24
WildSimplicialSurfaceExtensionByEdge-
Colouring
for IsSimplicialSurface and IsActualSurface and IsTriangleSurface, IsList, 23
WildSimplicialSurfaceExtensionByEdge-
ColouringNC
for IsSimplicialSurface and IsActualSurface and IsTriangleSurface, IsList, 23
WildSimplicialSurfaceExtensionBy-
Generators
for IsSimplicialSurface and IsActualSurface and IsTriangleSurface, IsList, 23
WildSimplicialSurfaceExtensionBy-
GeneratorsNC
for IsSimplicialSurface and IsActualSurface and IsTriangleSurface, IsList, 23